SANS ANALYST PROGRAM

Sponsored by Watchfire

Building Brick Houses

Applying Secure Lifecycle Practices to Web Applications

A SANS Whitepaper – October 2007

Written by: Gary W. Longsine and Jonathan Ham

House of Straw: Security as an Afterthought

House of Wood: Microsoft's Security Development Lifecycle (SDL)

House of Brick: Scalable & Agile Lifecycle Security for Applications (SALSA)

Summary

According to Symantec's Internet Threat Report, Web application exposures accounted for 77 percent of easily exploited new vulnerabilities in the last half of 2006. Although analysts may dispute the relevance of these simple metrics that merely count the number of exploitable holes, it is undeniable that Web applications are being exploited at an increasing rate. In many instances, they're being used as malicious installers or referrers to malicious installers of Trojan Horses – which made up 78 percent of newly-detected malicious code samples in August. Trojan Horses open back doors onto victim computers and download more code, most commonly turning those computers into Bots or Zombies. In June 2006, for example, it was widely reported that a Circuit City forum page was hacked and used to turn visiting computers into Spam-bots.

The attack methods being experienced as a result of new Web vulnerabilities have been around a while. SQL injections, cross-site scripting (XSS), cross-site request forgeries (XSRF), and forceful browsing have become popular again as new Web 2.0 technologies offer new attack surfaces for exploitation. Interactive applications that allow users to upload rich content hidden behind XML wrappers are opening new ways to send malicious content to a Web application via video, audio, and other files that today's security is not well prepared for. And vulnerabilities inherent in feature-rich, client-side applications, such as AJAX and Flash, are also being widely exploited.

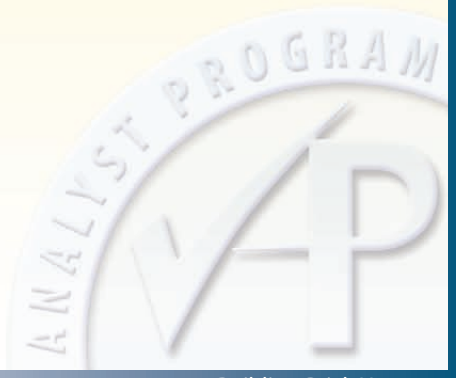
In addition to these new attack surface areas, IT organizations are under pressure to deliver competitive new services. This is pushing developers to work harder and faster, leaving fewer development resources to focus on items such as security, which are not explicitly driven by customer demand, and which customers don't realize they need until well into the application lifecycle.

Today, the wave of the wireless Web in your pocket, led by WiFi enabled devices like the Apple iPhone and iPod Touch, and by wireless 3G enabled devices like the Nokia N95, is greatly accelerating demand for efficient, globally distributed and powerful Web 2.0 applications in uncontrolled environments. There are over two billion cell phones in use today – 1.02 billion of them sold in 2006, according to an article in *The Inquirer* citing the Semiconductor Trade Associa-tion. These wireless users are eager to use the Internet on their handheld gadgets – not merely to "surf the web" and chat with friends, but also for shopping, banking, and accessing government services. Many organizations are rushing to fill the demand with innovative and performant solutions such as "Google Maps." As organizations rush to deliver Web 2.0 applications – selling tickets, making reservations in restaurants, buying and selling stock, and managing account balances – they'll be exposing more and more valuable enterprise and customer data to assault from the Internet.

As business lines began demanding more modern features in their Web applications, the technology to develop such applications improved to meet that demand. Application development tool-chains and function libraries such as Java, .Net, and Ruby on Rails were invented or evolved, leading to significant improvements in the ability of organizations to deliver new applications in less time. Drag and drop "Integrated Development Environments" evolved to support rapid Web services application development. Automated build systems and other "agile development" techniques were incorporated into the development processes in many organizations, even those using "traditional" or "waterfall" development methodologies.

By learning from the successful strategies employed in these prior evolutions of software technology, development teams can incorporate security elements directly into application lifecycle management to improve delivery and maintenance of applications. In this paper, SANS introduces a conceptual framework that provides a flexible approach to building and maintaining secure applications for Web 2.0 and other markets. We're calling this approach Scalable and Agile Lifecycle Security for Applications – or SALSA for short.

Compared to Microsoft's Software Development Lifecycle (SDL), which emphasizes reducing security errors in the design phase of an application lifecycle, SALSA seeks to provide additional cost-effective security best practices to address all areas of the application lifecycle: definition, design, development, testing, deployment, and management. The framework emphasizes scalable automated solutions where possible, and is modular and flexible enough to be incorporated easily into any existing software development process or methodology that organizations may already be using. SALSA is not intended to replace a methodology, but to open up dialog allowing organizations to explicitly consider security in the lifecycle management of their applications and application hosting environments.



House of Straw: Security as an Afterthought

Traditional software development methodologies, even most of the relatively modern "agile" Web development methodologies in use today, scarcely mention security. They almost universally consider application security (to the extent that it's consciously considered at all) to be one of the set of abstract application "requirements" to be defined, designed, developed, tested, deployed and managed along with all the other unique application requirements. The methodologies assume that your development teams understand how to gather, assess, prioritize, design, implement, test, and manage application security requirements, as they offer no special advice concerning these matters.

You might say that we built this first network of Web applications out of straw – and that house is being blown down around us because it is plagued by worms, botnets, spam, phishing attacks, and fraud.

House of Wood: Microsoft's Security Development Lifecycle (SDL)

Acknowledging these issues, and under tremendous pressure from enterprise and government customers and daily headlines about vulnerabilities in its ubiquitous products, Microsoft in 2002 began looking for ways to improve not only the security of its products, but also the processes used to build them with security as design intent.

One of the most visible and rigorously formalized approaches to security in the lifecycle of applications to date, the Security Development Lifecycle (SDL) marks a serious effort to incorporate security into the cradle-to-grave application lifecycle. The SANS whitepaper "Defining and Understanding Security in the Software Development Life Cycle," by James E. Purcell offers a brief overview of the major SDL phases. *The Security Development Lifecycle*, published by Microsoft Press, provides a detailed discussion of SDL, including some interesting background on the creation of the SDL itself.

SDL is particularly strong in the design phase of the software development lifecycle. SDL's Attack Surface Analysis and Attack Surface Reduction approaches, combined with developer training and a standard set of security "requirements" (often referred to as a "security checklist"), can be adopted by many kinds of development organizations. The SANS whitepaper "Building Security into the System Development Life Cycle (SDLC): A Case Study," by James E. Purcell offers an overview of SDL major phases and a pragmatic approach for organizations considering the wholesale adoption of SDL.

Appropriate and useful techniques can be adapted from SDL by leaner development organizations such as typical Web 2.0 development teams. However, most organizations will not be able to sustain the additional process weight that wholesale adoption of SDL imposes on application development, particularly its process-heavy "Final Security Review."

A new, lighter, more flexible framework for organizing application lifecycle security is needed to help facilitate sustainable and consistent development of secure applications.

House of Brick: Scalable & Agile Lifecycle Security for Applications (SALSA)

The key to scaling security to the coming wave of Web 2.0 applications lies in the enthusiastic adoption and wide application of best security practices in a framework provided by an agile development technique known as the automated build. By deliberately applying automated builds to the realm of application security, and to several phases of the application lifecycle, the major scalability challenges mentioned above can be overcome. In addition to providing sustainable security and quality improvements, the practice reduces the cost of repairing software defects, and reduces the cost of integration.

To achieve sustainable and scalable gains in application security, organizations must begin to incorporate best security practices into their Web application lifecycle management and software development methodologies. A SALSA approach emphasizes the automation of best security practices to the extent feasible, but also suggests other sustainable and scalable processes important to the security lifecycle of applications.

The SALSA framework recommends broad application of automated build technology, combined with automated security tools in the development, testing, deployment, and management of the application lifecycle. Automated processes of various types can reduce the potential for human error during the application lifecycle, reduce the effort required to enforce and verify security standards, and reduce security defects.

By themselves, none of the ideas suggested are unique. However, it is rare to see them employed in the manner suggested as part of a standard automated build system and otherwise highly automated processes for continuous integration or use.

Manage Your Enterprise Attack Surface

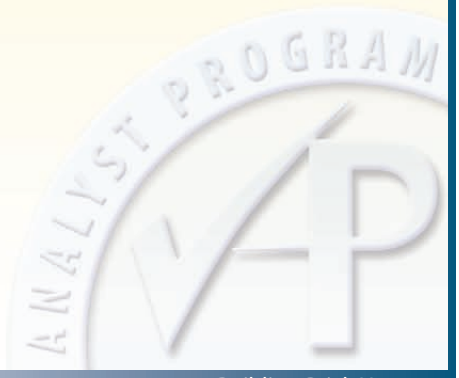
The sum of all exposed application interfaces constitutes the "enterprise attack surface." With thousands of dynamic Web pages hosted at several hundred physical servers globally, many enterprises don't even know what Internet-facing applications they have "floating around out there." These orphaned applications may not even have a maintenance developer responsible for the project. Conventional wisdom leads organizations to the mistaken impression that some of these applications are of little consequence from a security perspective because they do not directly expose data, record information from customers or other visitors, or represent a direct link back into the network. While it's critical to inventory, prioritize and manage applications transacting customer financial data, it's also important to realize that even the most benign sites can cause serious problems.

Just what happens when these unknown applications are not managed, patched and updated? They become increasingly threatened by evolving security threats. They may be loaded with a little i-Frame that relays visiting computers to malware sites, or relays malware to visiting computers. Your brand can be damaged when outside parties discover that your systems maliciously and effectively attacked the public without your knowledge. Applications may become vulnerable to defects discovered in browsers or other infrastructure components after deployment. Attacks like cross-site request forgery (XSRF) may exploit trust relationships between your application and the end user, brokered by the browser. Defects in a new version of a browser may lead to forged requests to act on behalf of that user without their knowledge, transferring funds for example.

The SALSA framework recommends regular use of the Attack Surface Analysis technique to help uncover issues like these during the design and deployment phases. How would you try to crack features and functions in your Web applications? Developers and implementers must think this way in planning, deployment, and throughout the lifecycle of the application.

Checklists should proscribe specific "worst practices" relevant to the environment or technologies in use, as well as require best practices such as use of the organization's standard directory for authentication. Design documents should include specific suggestions such as, "The application needs to store the customer name and social security number," rather than, "The application should use the customer social security number as an index in a relational data store."

Although there do not seem to be good opportunities for automation in the design phase, standard baseline security checklists can be quite useful in this phase, and might address this point by incorporating rules such as, "sensitive data including Social Security Numbers must not be used in ways that might lead to accidental exposure of that data," or "SSN must always be encrypted when stored in a database to safeguard against accidental exposure." SALSA also encourages the practice of having a brief feedback loop for review of definition and design when security defects are uncovered in any phase. Threat Modeling should be periodically conducted to help prioritize which items will be fixed sooner, and which later.



Integrate Security Best Practices into Your Application Lifecycle and Development Methodologies

As you review industry developments or as new practices emerge, identify those that can be useful to your organization and incorporate them into your processes. Attack Surface Analysis and Reduction can be incorporated as design tasks into design phases for any methodology. Security checklists of different kinds exist for different phases of the application lifecycle. Incorporate them into your standard process so that security items are performed consistently for each iteration, rather than as an afterthought.

Integrate Security Best Practices into Your Automated Build Process

In order to create a foundation for improving software quality and security, development organizations not yet doing automated builds must begin to apply automated processes to all outward facing applications, as well as Intranets handling sensitive employee data. An automated build system performs a suite of automated security checks on applications during development and production, something that should be done routinely throughout the application's lifecycle.

An Automated Build Process should include the following elements:

■ Metrics and Complexity limits

Several utilities can produce useful metrics on the codebase of software projects. Some of these, such tools as SLOCCode or JavaNCSS, may be generally useful to project managers. Some of these utilities, such as the Cyclomatic Complexity Number, also produce estimates of the complexity of given software modules. High complexity ratings in a given module may indicate a need for design review. Overly complex modules are more difficult to maintain, leading to accidental introduction of security flaws in design or coding of the module as the software evolves. By automating these assessments, alerts can be generated the same day that a module exceeds a given threshold, prompting a design review before the complexity of a given module spirals out of control. If a review uncovers ways to reduce the attack surface, for example, a code refactoring or design change can be scheduled for a future iteration.

Automated Code Analysis

Other utilities are available that analyze source code in a given language looking for particu-lar types of errors that can have security implications – commercial and open source tools are available that identify buffer overflows in C strings, for example. Identifying such tools for a given language and instrumenting them into an automated build process can reduce security defects and improve general code quality as well. Some of these systems can be extended to automate searchers for application-specific defects, once the first one has been discovered and analyzed.

Automated Unit Testing

Evolving projects should be instrumented with appropriate automated unit tests. As defects are encountered and fixed, automated tests should be generated to ensure that they stay fixed. This "off-the-shelf" practice from agile development methods is included here because the security posture of applications is generally improved by lower defect counts. Cascading defects can have unintended security consequences, accidentally exposing data for example, if a module crashes. When such defects are encountered, they should be fixed, and an automated test created to ensure that future changes do not break those fixes.

Automated Packaging

Application deployment packaging frequently must check or set permissions on files or make other system configurations. Automate the process of packaging to reduce human errors, which often introduce security defects into deployed systems at installation time. Automated packaging also improves the ability to deliver more frequent application updates to testing groups and customers.

The SALSA framework recommends automated packaging for commercial and in-house custom software. Daily build systems can also produce the installer package, allowing for more thorough testing of the deployment wrapper, a frequent source of install-time security defects, such as mangled ACLs. Automated packaging also facilitates other types of testing, both automated and ad-hoc manual, as it provides for easy rebuilding of a test environment. Test environments may then be easily equipped with Automated Fuzzing, and Automated Vulnerability Scanning, to greater effect, as new builds can be tested automatically in a simulated deployment environment.

■ Automated Fuzzing

In addition to the build-time tests, additional automated tests can be instrumented in the post-build phase using the same techniques and tools. One rapidly evolving such area involves "fuzzing," which means automatically generating large amounts of malformed data for given application interfaces. The intent is to discover defects that might confuse the application into granting unauthorized access, paralyze it into doing nothing when attack code slips in, or otherwise misbehaving in a way that is outside the design intent of the system. These techniques can discover defects in applications that might otherwise be very difficult to identify – until a malicious program finds and exploits them.

Automated fuzzing and automated scanning can identify defects in daily builds. The SALSA framework recommends incorporating these techniques into the automated build so that testing happens continuously, not at the end of a development cycle. This more efficient process moves defect discovery very close to the time of implementation, thus reducing the cost of identifying and correcting a defect later.

■ Automated Vulnerability Scanning

Various vulnerability scanners exist for assessing networks of deployed systems and applications. Some of the tests that these tools perform can help further reduce the defect rate in your application code and packaging. Following the build, automatically deploy to test machine in a secure test environment. Perform a full vulnerability scan on them, both before and after the deployment, and compare the results. Did your package accidentally open up a security hole in the operating system that had been previously closed? Yours wouldn't be the first.

☑ Integrate Security into Your Training and Cross-Training Plans

The SDL authors and other security experts have for years maintained that entire development and test teams should be trained in basic application security issues, and begin wrapping secure practices into standard development processes. Organizations have been slow to respond to this advice, preferring instead to rely on a small cadre of experts stretched too thin on reactive security issues. This slow progress is a contributing cause to the sad state of Web application lifecycle management that we find ourselves in today.

To improve the security of Web application development, SALSA recommends that all teams involved in the application lifecycle learn a common language for application security. Cross-training for business analysts who help define the application, developers who design and develop it, testers who look for defects, help desk workers who talk to the application users and systems administrators who monitor the application and support the other teams will be critical to security lifecycle success. Developers, testers and program managers must learn to understand the importance of security issues, communicate on those issues with others on the project, and take positive, proactive steps to reduce risks endemic to software applications.

The various groups involved in the different phases of the application lifecycle should also take a quick course on basic logic. Other IT communities that work with developers (Help Desk, Quality Assurance, Business Analysts who define requirements or Project Managers) could then use this as a common terminology as they grapple with everyday support issues, for example learning to recognize a security flaw in customer complaints or writing more effective help desk tickets.

Incorporate Continuous Process Improvement in Application Lifecycle Plans

Most organizations will benefit from regularly scheduled reviews of both high-level and detailed processes. Did new technologies emerge that could help automate additional functionality in the system? Did new threats emerge that require additional automated scanning tests? Is there a new version of your fuzzing tool available that understands more protocols you can take advantage of? The SALSA framework recommends that project architects ponder these high-level issues at least once a year, preferably once a quarter, picking the improvements to make, and scheduling them into future builds.

Deploy Automated Continuous Penetration Testing and Vulnerability Assessment

Reuse some of the work developed for the automated test environment to set up automated scans of your deployed application servers. If a configuration changes on a deployment server (either accidental, intentional but misguided, or nefarious), the problem may be detected early.

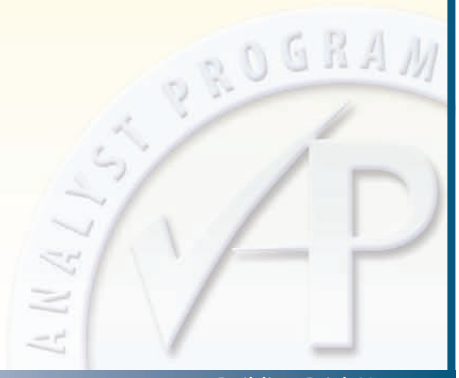
At a minimum, scan production servers for open ports each day (each hour if feasible) and generate alarms for open ports that shouldn't be open on a given host. The SALSA framework recommends starting with this initial automated scanning of the production environment. Once the foundation for this is in place, additional automated scans can be added, such as more advanced vulnerability scanning on particular ports. Be sure to provide for periodic updates of the scanning tools. They tend to evolve rapidly. As the scanning tools are updated, they may also uncover defects that previously remained undetected.

Improve Transparency Throughout the Application Lifecycle

The SALSA framework recommends improved transparency at every phase of the lifecycle of an application. Consider using a secure internal Web page as an entry point or portal to all phases of the lifecycle of an application, with links to definition phase requirements, design phase architecture documents, a running test application server, the bug reporting system, relevant e-mail lists and so forth. This simple step can improve participation and feedback, and help break down the communication barriers between different teams responsible for different phases of an application lifecycle.

Make it as easy as possible for your customers to report defects and submit enhancement requests. Include a "bug report" button in your application. Have it gather information from the running application or its log files to help you reproduce or diagnose bugs. Be sure to inform your users about the information your application is collecting, and to collect it via an encrypted tunnel so that it can't be sniffed by a third party.

Experience has shown that pathological security defects are often noticed first by the application users, so let them know how you're doing. The better your customers understand your basic application architecture, the better feedback they will be able to provide you.



Conclusion

The threat landscape for Internet-facing applications continues to accelerate as more applications are being demanded in harder-to-control environments such as cell phones and pocket PCs. Organizations need a scalable approach to improve the security posture and reduce the attack surface of the applications they build.

By extensively automating security best practices in different phases of the application lifecycle, security can be dramatically improved while at the same time staying competitive with the demand for more feature-rich Web 2.0 applications. SALSA (Scalable & Agile Lifecycle Security for Applications) is a suggested approach that adapts the principles, methods, techniques and tools from agile software development to improve the security of an organization's existing software development methodology or practices. In particular, the techniques and tools associated with automated builds and continuous integration can be applied to Web application lifecycles. The SALSA Framework is a practical, scalable approach that development teams can use to map their Web applications and develop new applications according to security lifecycle best practices.

About the Authors

Gary W. Longsine is the founder of Intrinsic Security, Inc. (http://intrinsicsecurity.com) where he brings uniquely effective anti-worm, anti-botnet, intrusion detection and intrusion suppression technology to enterprise, government and federal government organizations. He previously founded illumineX, Inc., and WebObjectsHosting.com, where he led software development teams using Agile development methods to produce scalable solutions in the areas of software distribution, large-scale urgent communications, commercial software license management, and a variety of dynamic interactive Web systems.

Jonathan Ham, a certified SANS instructor who teaches globally, holds CISSP, GSEC, GCIA, and GCIH certifications, and is a member of the GIAC Advisory Board. He is an independent consultant who specializes in large-scale enterprise security issues and who, for the past 12 years, has advised clients in both the public and private sectors, from small upstarts to the Fortune 500. He has been commissioned to teach NCIS investigators how to use Snort, performed packet analysis from a facility more than 2000 feet underground, and chartered and trained the CIRT (Computer Incident Response) for one of the largest U.S. civilian federal agencies. See his course schedule at http://jhamcorp.com.

Appendix I – Additional Resources

The Security Development Lifecycle
2006 Microsoft Press
Michael Howard and Steve Lipner
http://www.microsoft.com/mspress/books/8753.aspx

Software Security: Building Security In
2006 Addison-Wesley Software Security Series
Gary McGraw
http://www.amazon.com/Software-Security-Building-Addison-Wesley/dp/0321356705/ref=pd_sim_b_2/103-9964439-3385419?ie=UTF8&qid=1156801743&sr=1-1

Secure Coding: Principles & Practices
2003 O'Reilly
Mark G. Graff & Kenneth R. van Wyk
http://www.amazon.com/Secure-Coding-Principles-Mark-Graff/dp/0596002424/ref=pd_bbs_sr_1/103-9964439-3385419?ie=UTF8&s=books&qid=1190200432&sr=1-1

Gray Hat Hacking: The Ethical Hacker's Handbook
2004 McGraw-Hill/Osborn
Shon Harris, Allen Harper, et al.
http://www.amazon.com/Gray-Hat-Hacking-Ethical-Handbook/dp/0072257091/ref=pd_bbs_sr_1/103-9964439-3385419?ie=UTF8&s=books&qid=1190200560&sr=1-1

SANS would like to thank the sponsor